

أسم المقرر : هياكل البيانات

د/ مجدي ابراهيم الشرقاوي

المرجع :

Data Structures and Algorithms in C++

By

Adam Drozdek

Dr. Magdi Elmaghrabi

Classes

Each real-world object has its own properties and specific things that you can do with it.

A class is like a general model from which you can create objects. For example, if you create a dog class, you describe the characteristics related to the general idea of a dog.

Creating a class is the same as creating a new data type. When you create a class, you tell the computer the kind and amount of data this new type can hold. You also tell the computer what actions the new type can perform. Then you can use the class you've created to create variables from this new type (as you will see, these variables are called objects).

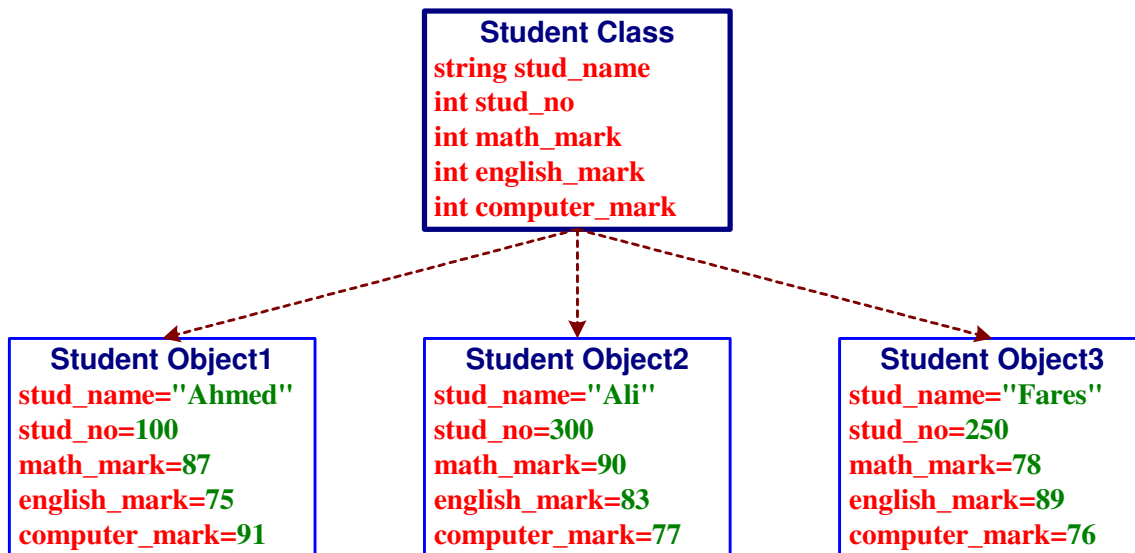
```
Class student
```

```
{
```

```
// fields and attributes – حقول البيانات
```

```
// methods – طرق ودوال معالجة هذه البيانات
```

```
}
```



Example:

```
class Ratio
{ public:
    void assign(int, int);
    double convert();
    void invert();
    void print();
private:
    int num, den;
};
```

The functions `assign()`, `convert()`, `invert()`, and `print()` are called *member functions* because they are members of the class. Similarly, the variables `num` and `den` are called *member data*. Member functions are also called *methods* and *services*.

```
class Ratio
{ public:
    void assign(int, int);
    double convert();
    void invert();
    void print();
private:
    int num, den;
};

int main()
{
    Ratio x;
    x.assign(22, 7);
    cout << "x = ";
    x.print();
    cout << " = " << x.convert() << endl;
    x.invert();
    cout << "1/x = "; x.print();
    cout << endl;
}

void Ratio::assign(int numerator, int denominator)
{
    num = numerator;
    den = denominator;
}

double Ratio::convert()
{
    return double(num)/den;
}

void Ratio::invert()
{
    int temp = num;
    num = den;
```

```
        den = temp;
    }
    void Ratio::print()
    {
        cout << num << '/' << den;
    }
}
```

شكل آخر للبرنامج

```
class Ratio
{ public:
    void assign(int n,int d)
        { num = n; den = d;}

    double convert()
        { return double(num)/den;}

    void invert()
        { int temp = num; num = den; den = temp; }

    void print()
        { cout << num << '/' << den; }

private:
    int num,den;
};
```

A *constructor* is a member function that is invoked automatically when an object is declared. A constructor function must have the same name as the class itself, and it is declared without return type. The following example illustrates how we can replace the `assign()` function with a constructor.

```
class Ratio
{ public:
    Ratio(int n,int d) { num = n; den = d; }
    void print() { cout << num << '/' << den; }
private:
    int num,den;
};

int main()
{
    Ratio x(-1,3), y(22,7);
    cout << "x = ";
    x.print();
    cout << " and y = ";
    y.print();
}
```

CONSTRUCTOR INITIALIZATION LISTS

Most constructors do nothing more than initialize the object's member data. Consequently, C++ provides a special syntactical device for constructors that simplifies this code. The device is an *initialization list*.

Here is the third constructor in the previous example, rewritten using an initialization list:

```
Ratio(int n,int d) : num(n), den(d) { }
```

The assignment statements in the function's body that assigned `n` to `num` and `d` to `den` are removed. Their action is handled by the initialization list shown in boldface. Note that the list begins with a colon and precedes the function body which is now empty.

Here is the `Ratio` class with its three constructors rewritten using initializer lists.

```
class Ratio
{ public:
    Ratio() : num(0),den(1) { }
    Ratio(int n) : num(n),den(1) { }
    Ratio(int n,int d) : num(n),den(d) { }
private:
    int num,den;
};
```

***p = *q - 1;**

	i	j	p	q	
	19	40	1080	1080	
	1080	1082	1084	1086	

(i)



q = &j;

	i	j	p	q	
	19	40	1080	1082	
	1080	1082	1084	1086	

(k)



***p = *q - 1;**

	i	j	p	q	
	39	40	1080	1082	
	1080	1082	1084	1086	

(m)



`int i = 15, j, *p, *q;`

	i	j	p	q
	15	?	?	?
	1080	1082	1084	1086

(a)

`p = &i;`

	i	j	p	q
	15	?	1080	?
	1080	1082	1084	1086

(b)



`*p = 20;`

	i	j	p	q
	20	?	1080	?
	1080	1082	1084	1086

(d)



`j = 2 * *p;`

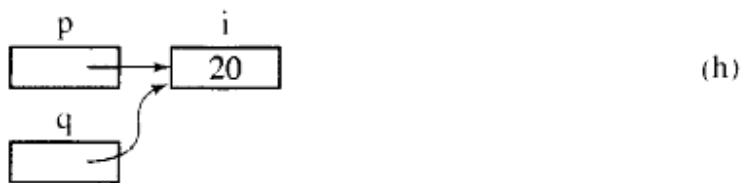
	i	j	p	q
	20	40	1080	?
	1080	1082	1084	1086

(f)

`q = &i;`

	i	j	p	q
	20	40	1080	1080
	1080	1082	1084	1086

(g)



Dr. May

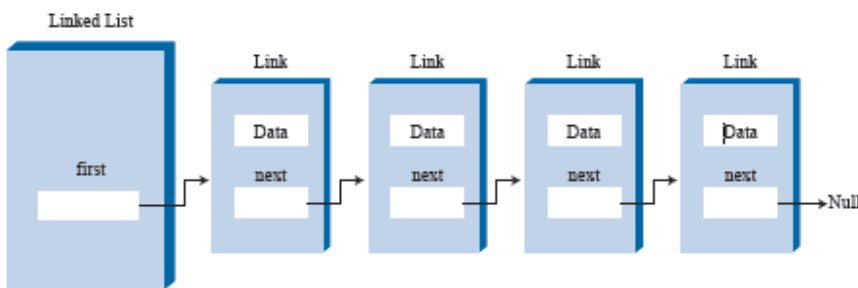
Linked Lists

An array is a very useful data structure provided in programming languages. However, it has at least two limitations:

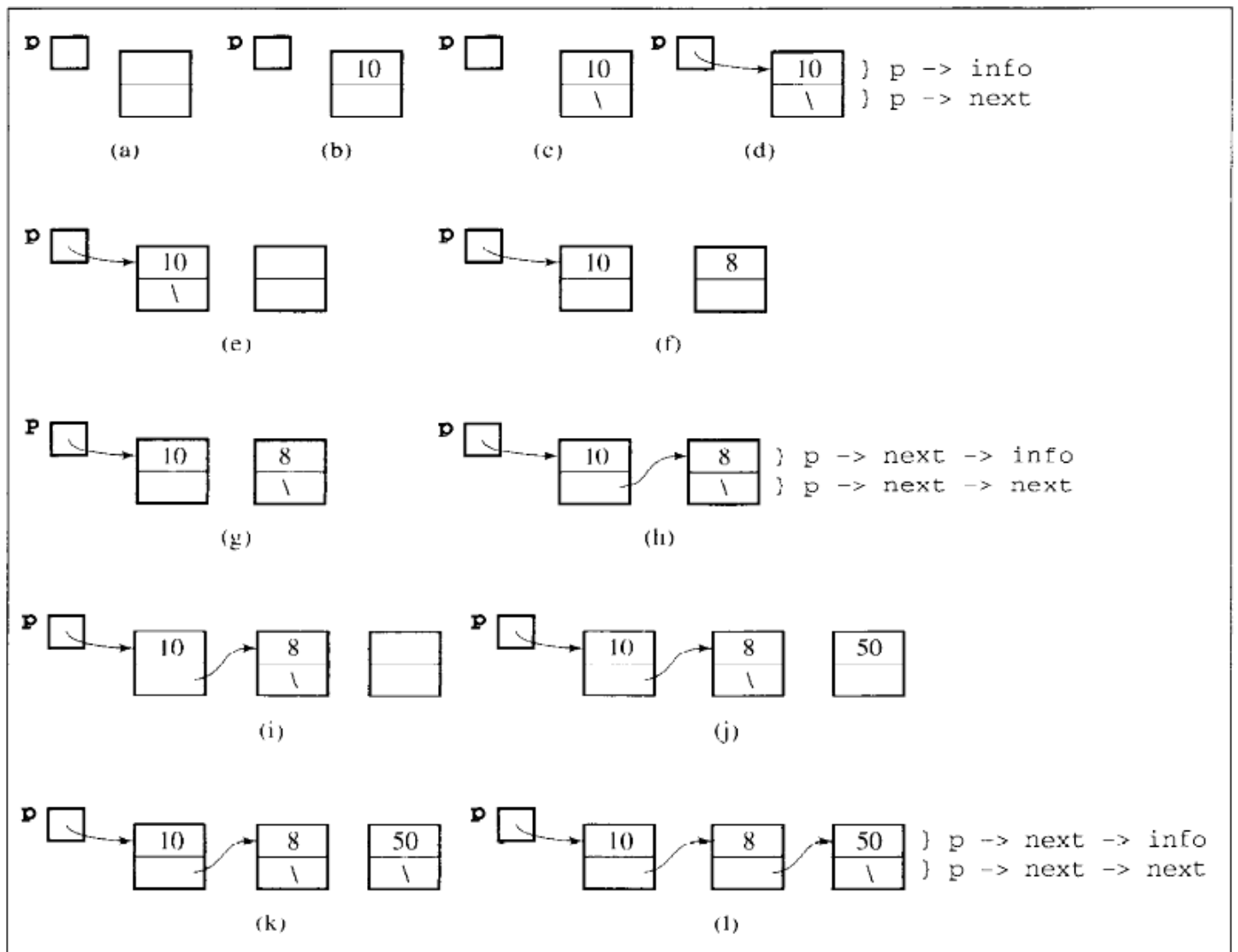
- (1) its size has to be known at compilation time
- (2) inserting an item inside the array requires shifting other data in this array.

This limitation can be overcome by using *linked structures*.

A linked structure is a collection of nodes located anywhere in memory, storing data and links to other nodes.



You can insert a new data item, search for a data item with a specified key, and delete a data item with a specified key.



```
#include "stdafx.h"
class IntNode {
public :
    IntNode() {next = 0; }
    IntNode(int i, IntNode *in = 0) {info = i; next = in;}
    int info;
    IntNode *next;
};

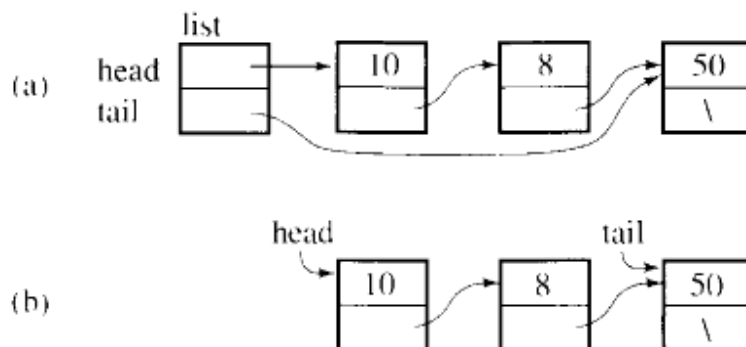
void main()
{
    IntNode *p = new IntNode(10);
    p->next=new IntNode(8);
    p->next->next= new IntNode(50);
    p->next->next->next= new IntNode(17);
}
```

```

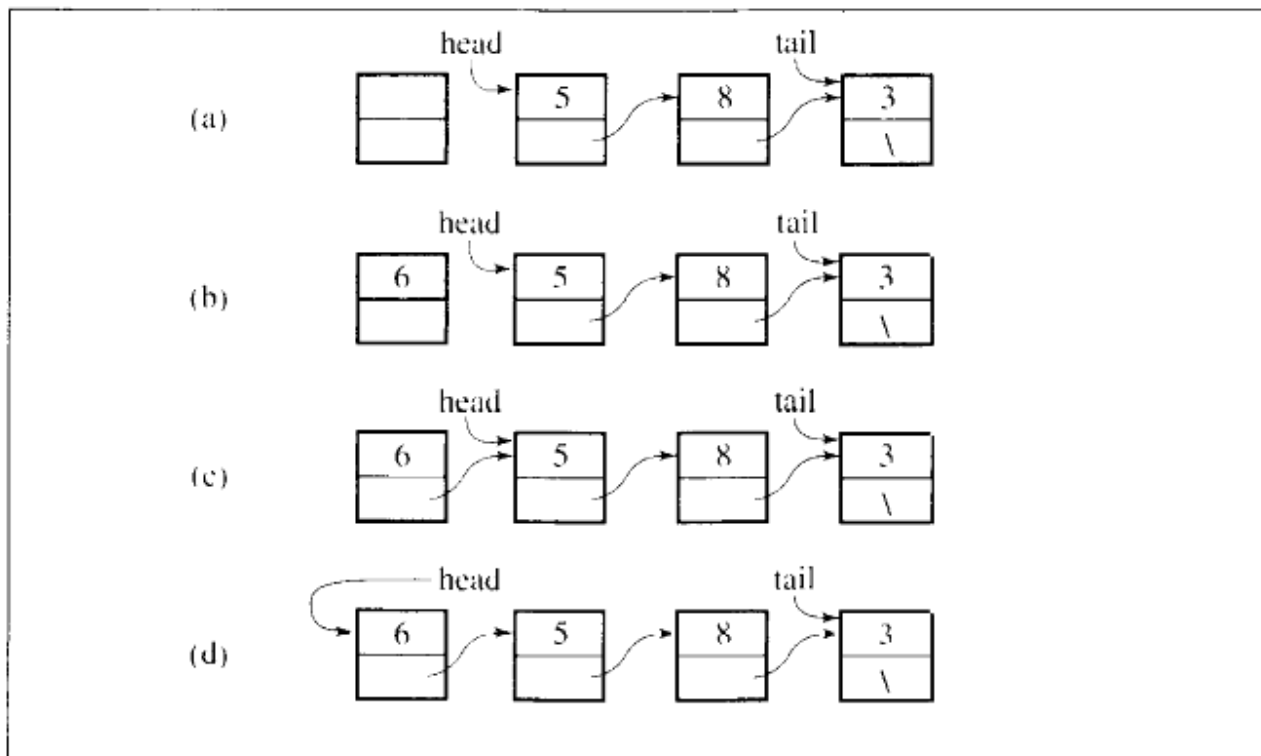
#include "stdafx.h"
#include <iostream>
using namespace std;
class IntNode {
public :
    IntNode() {next = 0; }
    IntNode(int i, IntNode *in = 0) {info = i; next = in;}
    void display() {cout << info << "\n";}
    int info;
    IntNode *next;
private :
    IntNode *head, *tail;
};

void main()
{
    IntNode *p = new IntNode(10);
    p->display();
    p->next=new IntNode(8);
    p->next->display();
    p->next->next= new IntNode(50);
    p->next->next->display();
}

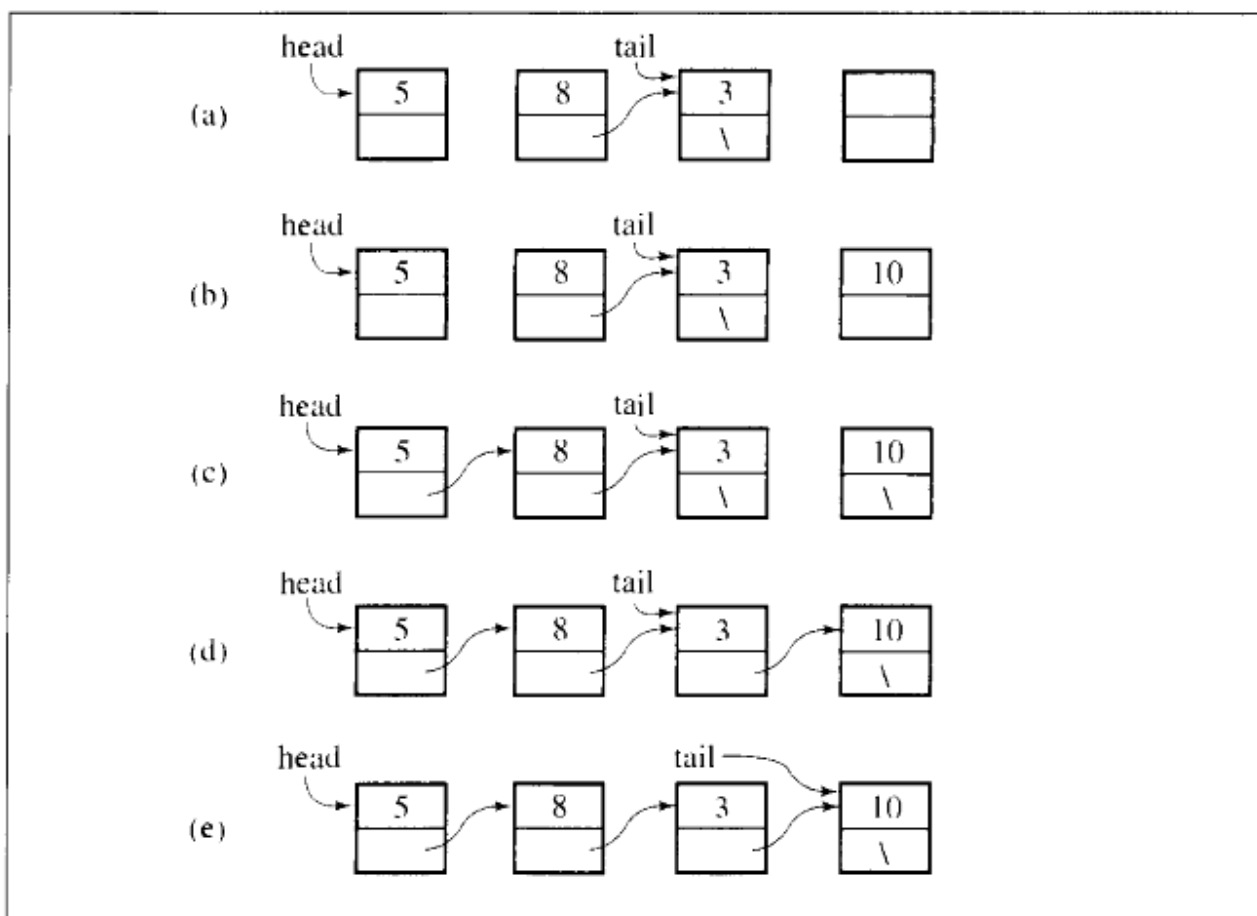
```



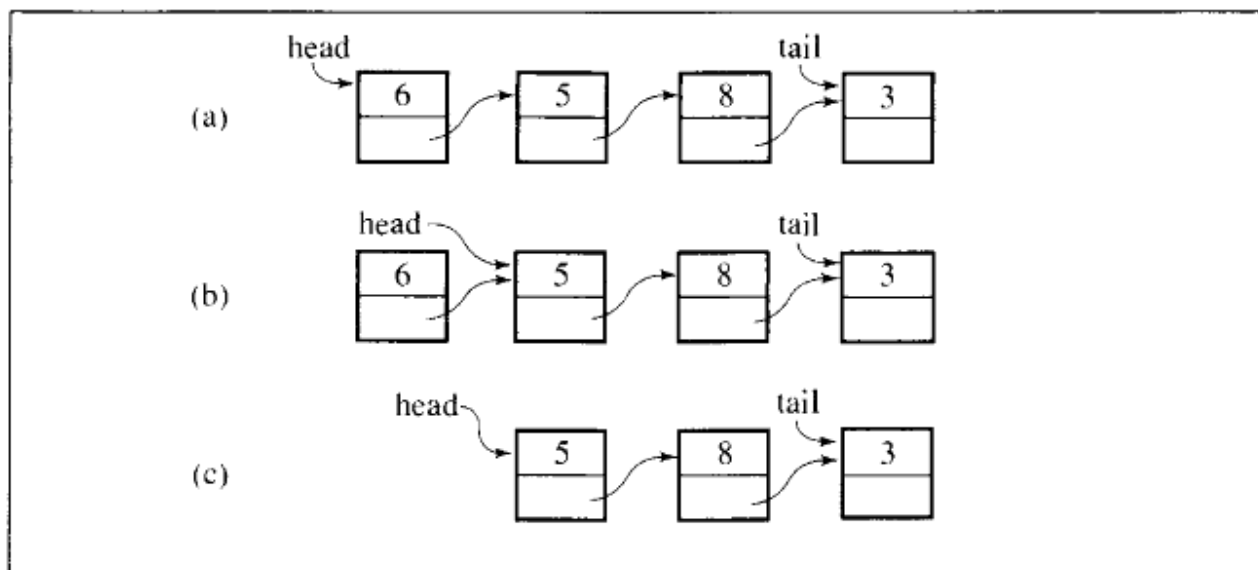
Inserting a new node at the beginning of a singly linked list.



Inserting a new node at the end of a singly linked list.



Deleting a node at the beginning of a singly linked list.



Deleting a node from the end of a singly linked list.

